logo, simge, sembol, yazı tipi, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

**TEKNOLOJİ FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ**

**GÖRÜNTÜ İŞLEME DERSİ**

**PROJE RAPORU**

**Yavuz Selim ŞAHİN – B200109031**

**Semih ÖZENÇ – B200109008**

**Yolo Algoritması ile Nesne Tespiti – Object Detection**

1. **Projenin Tanımı**

Görüntü işleme bir görüntünün içerisindeki önemli bilgilerin okunması, bu resimlerden anlam çıkartılması ve işlenmesi için kullanılan bir yöntemdir.

Nesne tespiti ise, bir görüntüde veya videoda bulunan nesnelerin tespit edilmesini ve belirlenmesini amaçlar. Bu nesneler, insanlar, arabalar, binalar, mobilyalar, yiyecekler, hayvanlar gibi çeşitli kategorilerde olabilir. Nesne tespiti algoritmaları, önceden eğitilmiş bir modeli kullanarak görüntü verilerindeki nesneleri tanımlar ve sınıflandırır.

Geleneksel nesne tespiti yöntemleri, görüntü işleme ve desen tanıma tekniklerini temel alır. Nesne tespiti ayrıca nesnelerin konumlarını da belirlemeyi hedefler. Bu, sınırlayıcı kutular (bounding boxes) veya piksel bazlı maskeler kullanılarak gerçekleştirilebilir. Nesne tespiti teknikleri, gerçek zamanlı uygulamalardan video analizine, otomasyon sistemlerine ve otonom araçlara kadar birçok alanda kullanılır. Özetle nesne tespiti, görüntü veya video verilerindeki nesneleri algılayan ve sınıflandıran tekniklerin kullanıldığı bir alanı ifade eder. Bu alandaki ilerlemeler, otonom sistemlerin gelişimi, güvenlik uygulamaları, nesnelerin takibi ve çeşitli endüstrilerdeki diğer uygulamalar için büyük potansiyel sunmaktadır.

Biz projemizde endemik bir bitki olan “Düzce Peygamber Çiçeği” adındaki çiçeği tespit edebilecek bir model geliştirmeye çalıştık. Bu çiçek Düzce’nin Gölyaka ilçesinde kırsal alanlarda bulunan ve koruma altında olan bir bitkidir. Ancak yerel halkın çoğunluğu bu çiçeğin böyle bir öneme sahip olduğunu bilmemektedir. Yapacağımız bu projeyle bu bitkinin tespiti ve endemik bir tür olduğunu kullanıcılara bildirilmesidir. Bu projenin bir amacı da ilerde tasarlanabilecek ve endemik bitkileri tanıyacak bir mobil uygulamada kullanılabilecek bir model tasarımıdır.

Projede Nesne Tespiti için YOLOv4 algoritması kullanılmıştır.

1. **Kullanılan Algoritma**

Yolo (You Only Look Once)

YOLO konvolüsyonel sinir ağları kullanarak nesne tespiti yapan bir algoritmadır. Açılımı ''You Only Look Once'' demektir. Sebebi ise algoritmanın nesne tespitini oldukça hızlı bir şekilde ve tek seferde yapabiliyor olmasıdır.YOLO algoritmasının diğer algoritmalardan daha hızlı olmasının sebebi resmin tamamını tek seferde nöral bir ağdan geçiriyor olmasıdır.

YOLO algoritması görüntüler üzerinde tespit ettiği nesnelerin çevresini bounding box ile çevreler.

YOLO kendisine girdi olarak verilen görüntüyü NxN’lik ızgaralara böler. Bu ızgaralar 5×5,9×9,17×17… olabilir.

Her ızgara kendi içerisinde nesne olup olmadığını ve nesne var olduğunu düşünüyorsa merkez noktasının kendi alanında olup olmadığını düşünür.

Nesnenin merkez noktasına sahip olduğuna karar veren ızgara o nesnenin sınıfını, yüksekliğini ve genişliğini bulup o nesnenin çevresine bounding box çizmelidir.

Bütün bounding boxların güven skoru vardır.

Birden fazla ızgara, nesnenin kendi içerisinde olduğunu düşünebilir. Bu durumda ekranda gereksiz bounding box’lar oluşur.

Birden çok bounding box olma durumunu engellemek için Non-Maximum Suppression algoritması kullanılır.

Kısaca Non-max Suppression algoritması görüntü üzerinde tespit edilen nesneler için çizilen bounding boxlardan güven değeri en yüksek olanı ekrana çizer.

araba, taşıt, araç, kara taşıtı, yol, karayolu içeren bir resim

Açıklama otomatik olarak oluşturuldu

Her bir ızgara kendi içinde, alanda nesnenin olup olmadığını, varsa orta noktasının içinde olup olmadığını, orta noktası da içindeyse uzunluğunu, yüksekliğini ve hangi sınıftan olduğunu bulmakla sorumlu. Daha açık anlatmak gerekirse örneğin yukarıdaki resimde arabanın orta noktası 7. ızgaraya denk geldiği için arabanın tespit edilmesinden/etrafına kutucuk çizmesinden o ızgara sorumlu.

**Güven skoru:** Bu skor modelin geçerli ızgara içinde nesne bulunup bulunmadığından ne kadar emin olduğunu gösterir. (0 ise kesinlikle yok 1 ise kesinlikle var) Eğer nesne olduğunu düşünürse de bu nesnenin gerçekten o nesne olup olmadığından ve etrafındaki kutunun koordinatlarından ne kadar emin olduğunu gösterir.

**Eğitim İşleminin Başlaması için gereken bazı dosyalar**

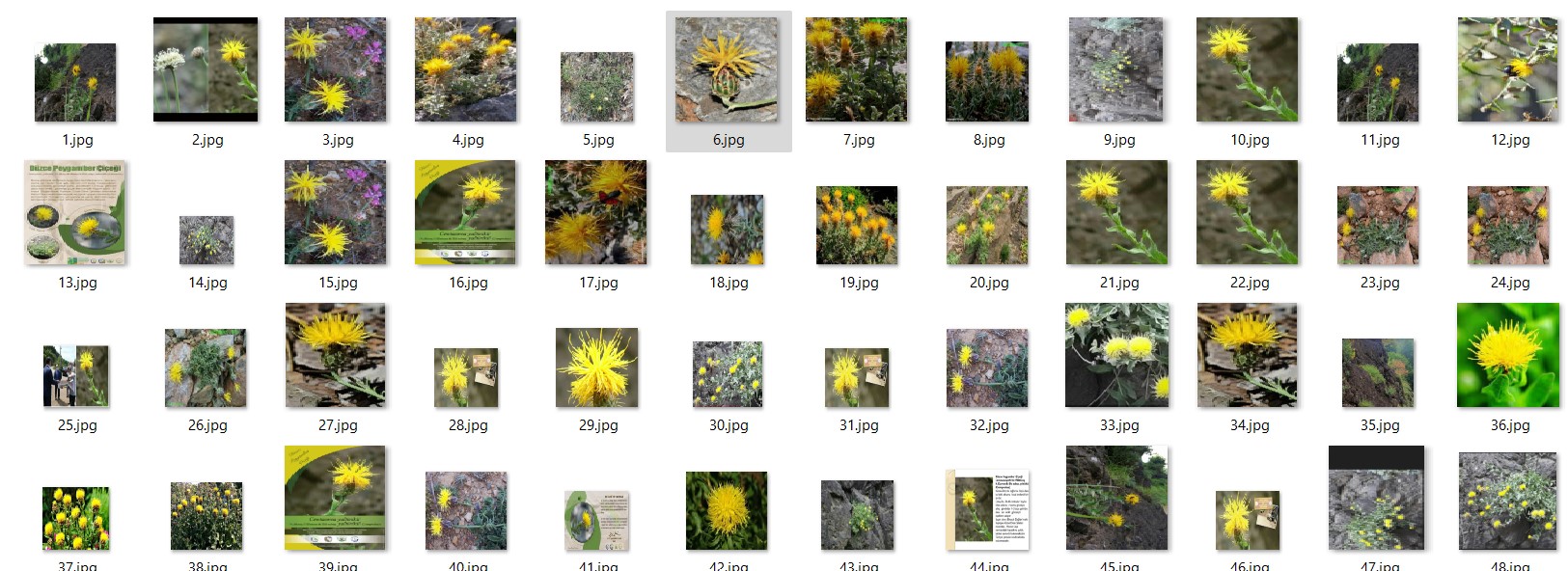
* YOLO algoritmasının çalıştırılabilmesi için Darknet bizden bazı dosyalar beklemektedir. Bunlardan bir tanesi de .cfg dosyasıdır.
* Config dosyası YOLO içerisindeki Sinir ağının başarısını,hızını vs etkileyecek özellikleri bizden talep eder.
* Train.txt , test.txt dosyaları
* .data dosyası . Gereken dosyaların konumlarını barındırır.
* YOLO’nun bizden beklediği bir başka dosya ise .names dosyası.Names içerisinde sınıfların adlarını tutar.
* Bir eğitimin başarısını eğitim sonrası oluşan loss grafiğinden anlaşılabilir.
* Eğitim aşamasını tamamlayıp .kabul edilebilir bir loss değere sahip .weights dosyasını elde ettiğimizde duruyoruz. Artık modelimiz hazır.

1. **Uygulama Adımları**

**3.1) Veri Setinin elde edilmesi**

Yapılacak proje için daha önceden hazırlanmış bir veri seti olmadığı için veri setimizi kendimiz oluşturduk. Veri setini hazırlamadan önce Düzce Üniversitesi’nde bitkiyi keşfeden öğretim görevlisiyle, proje hakkında bilgi verildi ve bitkinin resimleri için iletişime geçildi ancak herhangi bir dönüt alınamadı. Bu yüzden veri setini kendimiz oluşturmaya çalıştık. Veri setini oluşturmak için internetteki resimler, üniversitelerin herbaryumları , videolardan ekran kesintileri. Ancak 50 kadar resim toplana bildi. Bunun üzerine eldeki verileri artırmak için resim döndürme vb. yöntemler kullanılarak 174 resime ulaşıldı.

Resimde hazırlanmış veri setinden bir kesit gösterilmektedir.



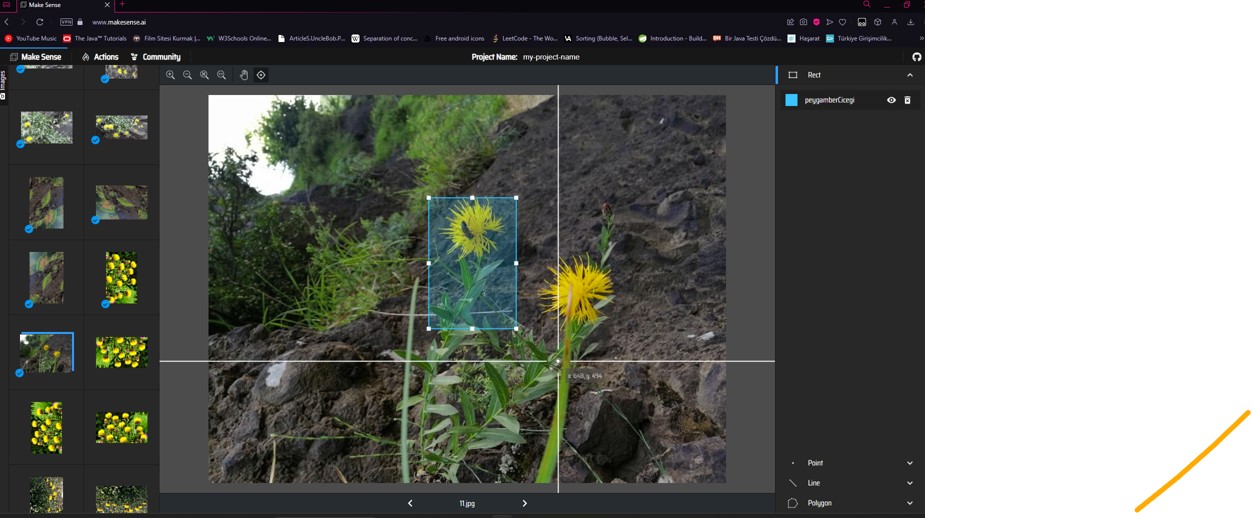
**3.2) Resimlerin Etiketlenmesi**

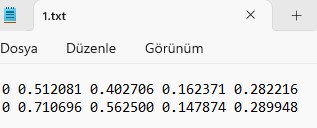
Resim etiketleme işlemi, bir görüntünün içeriğini tanımlamak veya belirli özelliklerini etiketlemek amacıyla kullanılan bir süreçtir. Bu işlem genellikle makine öğrenimi ve yapay zeka teknikleri kullanılarak gerçekleştirilir.

Resim etiketleme, bilgisayarın resimlerdeki nesneleri, sahneleri veya diğer özellikleri tanımasını ve bunları doğru şekilde etiketlemesini sağlar. Örneğin, bir resimdeki insanları, arabaları, hayvanları veya doğal manzaraları tanımlamak için etiketleme yapılabilir. Etiketler, resimdeki nesneleri veya özellikleri tanımlamak için genellikle kategorik veya açıklayıcı terimlerdir.

Projemizde resim etiketleme işi için “makesense.ai” sitesini kullandık. Resimleri etiketledik sonra YOLO formatında indirdik. Bu .txt uzantılı dosyalarda etiketlenen resimlerdeki nesne tespitinde kullanılacak konum bilgileri ve etiket bilgileri yer almaktadır.

Etiketleme işlemleri gösteren görseller.





1.txt uzantılı dosyada , 1.jpg resmi ile ilgili bitkinin konumu ve özelliklerini gösteren bilgililer.

**3.2) Darknet ‘ in elde edilmesi ve dosya düzenlemeleri**

Github’dan Darknet dosyalarını git clone ile indiriyoruz . İçinde YoloV4 algoritmasını barındırmaktadır.

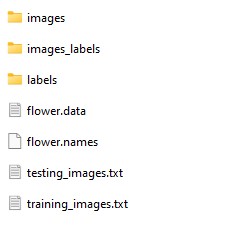
Ardından eğitim işlemine başlayabilmemiz için bazı düzenlemeler yapmamız gerekiyor.

* 1. images klasörü = içinde 174 tane resim ve Yolo formatında etiketlediğimiz dosyalar bulunuyor.
  2. Labels klasörü = İçerisinde her resme için Yolo formatında çıktı bulunuyor
  3. flower.names = Buraya mevcut sınıflarımızı yazıyotuz.
  4. Training.txt = Eğitim verilerinin konumu bulunuyor. 1.jpg , 2.jpg. 3.jpg …..48.jpg (174 tane resmin %80 i burda bulunuyor.)
  5. Testing.txt = Test verilerinin konumu bulunuyor. 48.jpg , 49.jpg ,50.jpg….

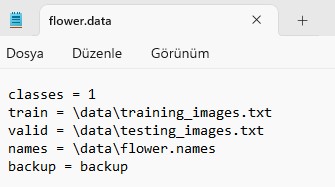
.60.jpg (174 tane resmin %20 si burada bulunuyor.)

* 1. flower.data = test , train , images , data ve ağırlık dosyalarının kaydedileceği konumları barındıran dosya.

Resimde hazırlanmış olan gerekli dosyalar gösterilmektedir.



Flower.data dosyasının içeriği .



**3.2) Config ve Makefile dosyalarının düzenlenmesi**

Darknet’in içindeki config ve Makefile dosyalarını eğitime başlamadan önce düzenlememiz gerekiyor.

Google Colab ve GPU ve OPENCV kullanacağımız için Makefile içerisindeki

GPU = 0 GPU = 1

CUDNN=0 CUDNN = 1

OPENCV=0 bilgilerini OPENCV = 1 olarak değiştiriyoruz.

Ardından yolov4.cfg dosyasını düzenleyeceğiz. Burada YOLO’nun verilerimizi train edebilmesi için özel ayarlar bulunuyor. Bu dosyanın içine baktığımızda batch ve subdivisions değerlerini görebiliriz. Batch her iterasyonda kaç resim üzerinden geçileceğini gösterir. Subdivision ise batchleri mini-batch gruplarına ayırır. Ayrıca bu **.**cfg uzantılı konfigürasyon dosyamızda her bir YOLO katmanındaki classdeğişkeninin değerini train edeceğimiz sınıf sayısı ile değiştiriyoruz ve her bir Yolo katmanının üzerindeki katmanda filtersdeğerini class sayımıza göre değiştiriyoruz.

**3.3) Dosyaların Colab’a yüklenmesi ve Eğitim İşlemi**

Veri setimizden elde ettiğimiz 6 dosyayı , Github’dan elde ettiğimiz darknet dosyamızın içine atmıştık. Darknet dosyamızı zipleyip Google Drive’a atıyoruz ve Google Colab ile Drive bağlantımızı sağlıyoruz.

Google Colab kullanmamızın nedeni Google’nin sağladığı Güçlü bir GPU olduğu için eğitim işlemi çok daha hızlı sürüyor.

**Colabtaki bazı komutlar.**

!unzip "/content/drive/MyDrive/Flower\_Detection/darknet.zip" Attığımız zip dosyasını çıkarıyoruz

%cd /content/darknet Dosyamızın içine giriyoruz.

!find . -type f -print0 | xargs -0 dos2unix Dosyaları Unix formatına çeviriyoruz

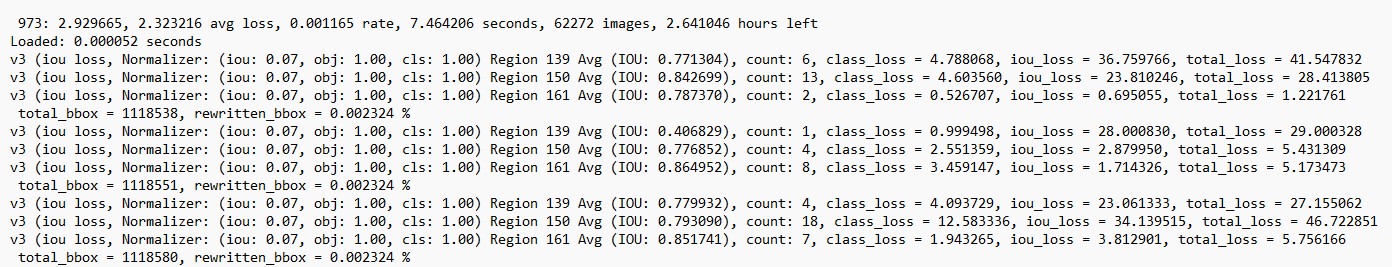
!chmod +x /content/darknet Gerekli izinleri alma işlemi

!make Makefile dosyamızı çalıştırıyoruz.

!./darknet detector train Flower\_Data/flower.data yolov4.cfg yolov4.conv.137 -map -dont\_show Eğitime başlıyoruz. Hazırladığımız verilerdeki flower.data , düzenlediğimiz yolov4.cfg dosyaları ve daha önce hazırlanmış ağırlık dosyası olan yolov4.conv.137 dosyalarını kullanarak eğitime başlıyoruz. Daha önceden eğitilmiş bir ağırlık dosyası kullanmamızın nedeni eğitime 0’dan başlarsak eğitimin çok uzun sürebileceğindendir.

**3.4) Eğitim İşlemi**

Eğitime başladıktan sonra bizim belirlemiş olduğumuz train ve test resimlerine göre modelimizin öğrenmesi de başlıyor. İlk başta 2000 loss değeri ile eğitime başlıyoruz. Süre geçtikte modelimiz öğreniyor ve loss değeri düşüyor. Loss değerimizin düşük olması modelimizin nesneyi tespit etmede o kadar başarılı olduğu anlamına gelir. Genel olarak 1’in altında bir loss değeri kullanışlıdır.



Örnek verilen resimde 973. Adımdaki bilgiler yer almaktadır. 1. Adımda 2000 olan avg loss değerlerinin 2.323 e kadar düştüğü gözlemlenebilir.

Hedefimiz 1’in altında loss değerleri elde etmek ancak loss değerlerinin aynı değerlerde takılı kalması (daha fazla öğrenememe) gibi durumlarla karşılaşabiliriz. Böyle bir durumda eğitim durdurulup eldeki ağırlık dosyası alınabilir. Eldeki ağırlık dosyası yeterli gelmediyse train ve test verilerinin güncellenip genişletilmesi elzemdir.

"avg loss" ortalama kaybı ifade eder ve genellikle bir modelin eğitimi sırasında hesaplanır. Kayıp, modelin görevinde ne kadar iyi performans gösterdiğini gösteren bir ölçüdür. Tahmin edilen çıktı ile gerçek çıktı arasındaki farkı temsil eder.

"rate" genellikle makine öğrenimi bağlamında öğrenme hızını ifade eder. Öğrenme hızı, modelin parametrelerinin eğitim sırasında güncellendiği adım büyüklüğünü belirler. Modelin eğitim verilerinden ne kadar hızlı veya yavaş öğrendiğini kontrol eder.

"seconds" belirli bir iterasyon veya adımın eğitim sürecinde geçen süreyi ifade eder. İlgili adımın süresini saniye cinsinden gösterir.

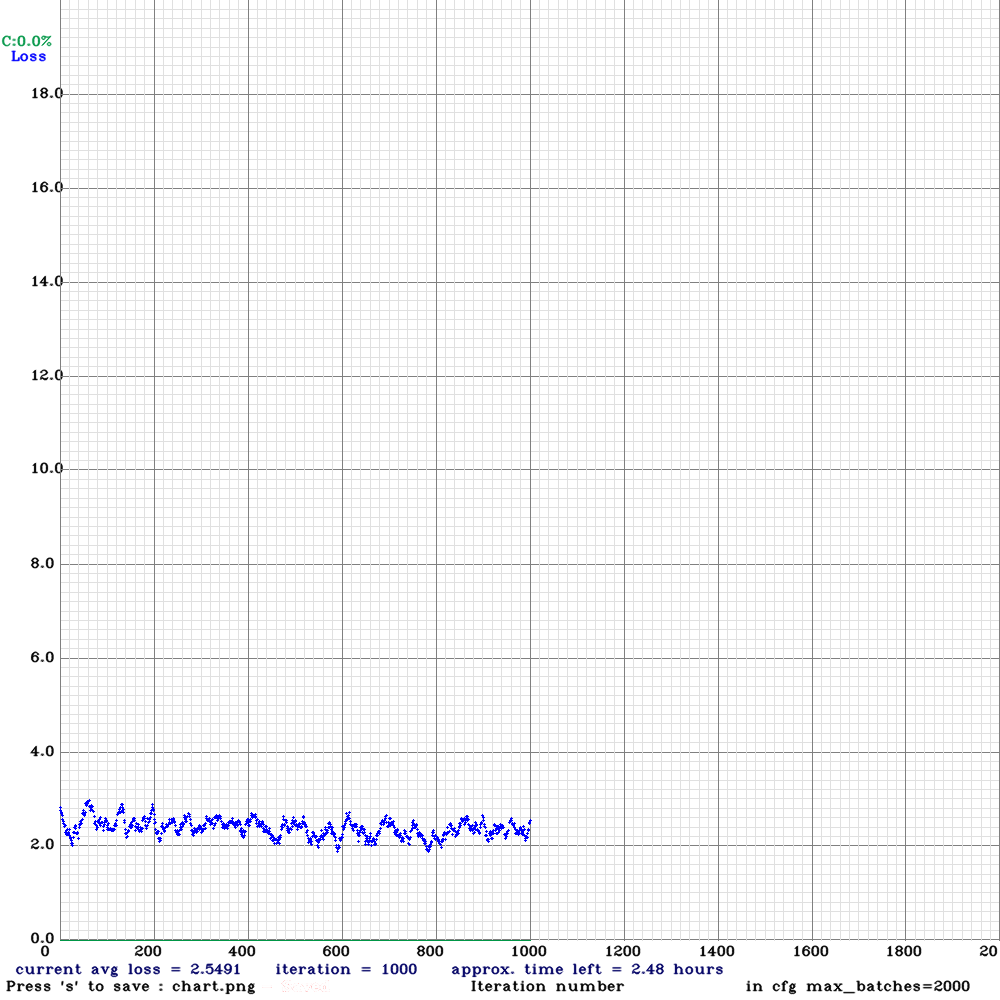
"images" belirli bir noktaya kadar işlenen veya eğitimde kullanılan görüntülerin sayısını ifade eder. Veri kümesinin boyutunu veya modelin eğitimi için kullanılan örnek sayısını temsil eder.

Mavi işaretlerle modelimizin öğrenme durumunu görüntüleyebiliriz. İlk aşama öğrenme grafiği şekildedir.

metin, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, çizgi, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu grafikte eğitimimi sonlandırıyoruz . Ve .weights uzantılı dosyamızı alıyoruz. Modelin eğitimine bu ağırlık dosyasından devam edeceğiz.



2. grafikte kayda değer bir ilerleme belirli noktadan sonra sağlanamamıştır. Bunun nedenini elimizdeki veri setinin kısıtlı olmasından dolayı olduğunu düşünüyoruz. Modelin daha fazla öğrenemediği görülmektedir . Optimum avg loss değeri 1.71 olarak ölçülmüştür. Eğitimi burada sonlandırıp model test aşamasına geçiyoruz.

**3.5 ) Modelin Denenmesi**

Test için eğitimler sonucunda elde etmiş olduğumuz ağırlık dosyası (.weitghs uzantılı dosya) ve config dosyamız gerekiyor. Ayrıca test edeceğimiz ortam için Spyder kullandık. Spyder içinde bir Python dosyası açtık ve opencv kütüphanemizi import ettik. Birden fazla bounding box oluşması durumunun önüne geçmek içinde sadece en yüksek skorlu bounding boxun oluşturulması için Non-Maximum Supression yöntemi kullandık

Python kodu : (Kod kısmı mor yazı ile verilmiştir)

"""

@author: YAVUZ

"""

import cv2

import numpy as np

img = cv2.imread(r"C:\Users\YAVUZ\Desktop\tespit\1.jpg")

#print(img)

#%% 2. Bölüm

img\_width = img.shape[1]

img\_height = img.shape[0]

# görüntüyü blob formata çeviriyoruz : 255

#(değişken, sabit değer, yolo416 blob ölçeği, bgr-rgb değişimi, crop)

img\_blob = cv2.dnn.blobFromImage(img, 1/255, (416,416), swapRB=True, crop=False)

labels = ["Peygamber Cicegi"]

# modelin tanıyacağı labelları giriyoruz :

# kutucuk renkleri ayarlıyoruz :

# buradaki kodları teker teker konsola yazınca değer oluşuyor

colors = ["255, 0, 0"]

colors = [np.array(color.split(",")).astype("int") for color in colors]

colors = np.array(colors)

colors = np.tile(colors,(30,1))

#%% 3. Bölüm

model = cv2.dnn.readNetFromDarknet(r"C:\Users\YAVUZ\Desktop\tespit\yolov4.cfg",r"C:\Users\YAVUZ\Desktop\tespit\yolov4\_lasto.weights")

layers = model.getLayerNames()

output\_layer = [layers[layer-1] for layer in model.getUnconnectedOutLayers()]

model.setInput(img\_blob)

detection\_layers = model.forward(output\_layer)

# ================================ non-maximum supression : operation 1 ================================

# fazladan kutuları yok etmek için kullandık . Sonuçta en yüksek oranlı kutuyu gösterecek :

ids\_list = []

boxes\_list = []

confidences\_list = []

# ================================ non-maximum supression : operation 1 end =============================

# deteksiyona başlıyoruz. for içinde for yaparak değerler oluşturduk

for detection\_layer in detection\_layers:

for object\_detection in detection\_layer:

scores = object\_detection[5:] # puan tutuyoruz, 5 değer aldık

predicted\_id = np.argmax(scores) # en yüksek değerli indeksi çekiyoruz

confidence = scores[predicted\_id] # en güvenilir skoru alıyoruyz ve tutuyoruz

if confidence > 0.2: # güven skoru bu değerlerden büyükler için işlem devam edecek

# kutuyu çizerken sol alt köşeden başlayıp sağ üst köşeye gideceğiz

label = labels[predicted\_id]

bounding\_box = object\_detection[0:4] \* np.array([img\_width,img\_height,img\_width,img\_height])

(box\_center\_x, box\_center\_y, box\_width, box\_height) = bounding\_box.astype("int")

# x ve y noktalarının özel koordinatı

start\_x = int(box\_center\_x - (box\_width / 2))

start\_y = int(box\_center\_y - (box\_height / 2))

#==================== non-maximum supression : operation 2 ====================

# döngü içindeki değerleri listeliyoruz ve alta iletiyoruz

# for içinde yukarda oluşturulan kutuları dolduruyoruz :

ids\_list.append(predicted\_id)

confidences\_list.append(float(confidence))

boxes\_list.append([start\_x, start\_y, int(box\_width), int(box\_height)]

# 0.5 ve 0.4 trashold değerleri yani standart

max\_ids = cv2.dnn.NMSBoxes(boxes\_list, confidences\_list, 0.5, 0.4)

for max\_id in max\_ids: # liste içindeki değeri çekeceğiz

max\_class\_id = max\_id # max\_class\_id aslında nesnenin tutulduğu id olacak

box = boxes\_list[max\_class\_id] # box en iyi değeri tutacak

start\_x = box[0] # box'un başlangıç noktası indis değeri

start\_y = box[1]

box\_width = box[2]

box\_height = box[3] # box eni ve boyu

predicted\_id = ids\_list[max\_class\_id]

label = labels[predicted\_id] # ilgili labelı yukardan uygun şekilde çektik

confidence = confidences\_list[max\_class\_id] # confidence oranı sağlam olanı aldık

end\_x = start\_x + box\_width

end\_y = start\_y + box\_height

box\_color = colors[predicted\_id]

box\_color = [int(each) for each in box\_color]

label = "{}: {:.2f}%".format(label, confidence \* 100)

print("Endemik Bitki Tespiti {}".format(label))

# kutuyla ilgili tüm parametreler hazır. çizim başlıyor :

cv2.rectangle(img, (start\_x,start\_y),(end\_x,end\_y),box\_color,2)

cv2.putText(img,label,(start\_x,start\_y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 1, box\_color, 3)

cv2.imwrite("tespit.jpg", img) #bize resim dosyası olarak çıktı verecek

**3.6) Ekran Görüntüleri**











Bitkinin doğadaki hali ve Modele gösterildikten sonraki çıktısı.



**KAYNAKÇA**

1. https://medium.com/@gizemcumen85/görüntü-i̇şleme-teknolojisi-image-processing-262bb58fbb27
2. <https://bulutistan.com/blog/goruntu-isleme/>
3. <https://tr.wikipedia.org/wiki/Görüntü_işleme>
4. <https://ece-akdagli.medium.com/yolo-algoritması-nedir-859862c7d5a6#:~:text=YOLO%20(You%20only%20look%20once)%20türkçe%20karşılığı%20%27Yalnızca%20bir,CNN%20kullanan%20en%20yaygın%20algoritmadır>.
5. <https://www.odakarge.com/yolo-nedir.html>
6. https://smartera.com.tr/gercek-zamanli-nesne-takibireal-time-object-detection-w-yolo-python/#:~:text=YOLO%20konvolüsyonel%20sinir%20ağları%20kullanarak,ve%20tek%20seferde%20yapabiliyor%20olmasıdır.